Package 'AssFunc'

May 13, 2016

Type Package

Title What the package does (short line)

Version 1.0

Date 2016-04-25

Author Connor F. White

Maintainer `<Connor.white@gmail.com>`

Description Assorted functions with timeseries data

License What license is it under?

R topics documented:

AssFunc-package *Assorted Functions ~~ AssFunc ~~*

Description

Functions writted during my Masters at CSULB. Originally designed for use with acoustic telemetry data and time series data. Used for generating and plotting transistion matrices. As well as identifying events in data.

Details

Package:	AssFunc
Type:	Package
Version:	1.0
Date:	2016-04-25
License:	What license is it under?

Focus on transMat for generating transistion matrixes and transBub for displaying the matrix

events can be used for identifying the start and end of a string of events. Designed for identifying individual dives from depth data, but can be used to identify any sort of event in time series data. functions can be applied to each event using eventFunc, or if NAs can be interpolated with eventInterp

Sunrise and suset times can be calculated using sunCalc

The spackage also contains some spatial analysis such as focal analysis on a raster using focalfast, converting coordinates to a cartesian frame using lat2cart, or calculating distances between two points that are in latitude and longitude

Author(s)

Connor F. White

Maintainer: <connor.white@gmail.com>

References

~~ Literature or other references for background information ~~

See Also

~~ Optional links to other man pages, e.g. ~~ ~~ <pkg> ~~

Examples

```
dat<-sample(c("A","B","C","D"),100,replace=TRUE)
x<-transMat(dat, States=c("A","B","C","D"))

transBub(x)


x<-c(.5,.5,.75,.25,.2,.25,.6,.9,.1,.7)
```

```
x.e<-event(x<.3)
```

---

bend *To Bend lines*

Description

Used in order to interpolate between points so that teh path of the line will be bent and not straight

Usage

```
bend(loc1, loc2, bend = 0.1, n=100)
```

Arguments

`loc1`	vector of length 2 of the first location
`loc2`	vector of length 2 of the second location
`bend`	porportion of distance between loc 1 and loc 2 to bend the line
`n`	number of points

Value

This function returns a matrix in which the first column is the x coordinates of n points, while the second column is the y coordinates

Note

Used to bend lines between two points, to reduce overlap

Author(s)

Connor F. White

See Also

`transBub`

Examples

```
first<-c(1,1)
second<- c(2,3)

l<-bend(first,second,bend=.1)

plot(first[1],first[2],xlim=c(0,3),ylim=c(0,4))
points(second[1],second[2])
lines(l[,1],l[,2])
```

bubcex *Scaling Bubble sizes*

Description

Based on some input, rescale the variables so that they have these cex sizes

Usage

```
bubcex(bubCounts, bubsize = c(1, 5), lims.b = NULL, sqrt = FALSE)
```

Arguments

`bubCounts`	Count data representing the counts in each bubble
`bubsize`	a vector of two showing the minimum and maximum sizes you want
`lims.b`	a vector of length two representing the min and max value things will be scaled to
`sqrt`	TRUE/FALSE whether you would like the vector to be squareroot transformed

Value

returns a vector the same length as bubCounts representing the cex of the bubbles

Author(s)

Connor F. White

See Also

`transBub`

Examples

```
counts<-rpois(10, lambda=5)

cexs<-bubcex(bubCounts=counts)
```

circLocs — *To Generate Locations on a Circle*

Description

Generating locations on a circle

Usage

```
circLocs(n,d=1)
```

Arguments

`n`	number of points on the circle you would like to generate
`d`	The diameter of the circle

Value

Returns a matrix with a column representing the the number of the circle, a column representing the Y coordinate and a column representing the x coordinate

Author(s)

Connor F. White

See Also

`transBub`

Examples

```
locs<-circLocs(n=10,d=3)

plot(locs[,2],locs[,3])
```

countProb — *Transision counts into probabilities*

Description

Turning Transisiton matrix counts into state transition probabilities

Usage

```
countProb(transMat)
```

Arguments

`transMat`	an n by n matrix with count data

Details

t

Value

Returns an n by n matrix, with the columns representing the current state and the rows representing the probability of transitionf to that state. Columns will sum to 1

Author(s)

Connor F White

See Also

transMat

Examples

```
rec<-sample("A","B","C",size=200)
trans<-transMat(rec)

probMat<-countProb(trans)
```

`distGeo` *Calculate Distance*

Description

calculate distance beteen lat and long

Usage

```
distGeo(lat, long, latOrg, longOrg)
```

Arguments

`lat`	Latitude of point
`long`	longitude of point
`latOrg`	latitude of second point
`longOrg`	longitude of second point

Value

the number of meters between the two points

Author(s)

Connor F. White

See Also

lat2Cart

Examples

```
disGeo(33,-118,34,-118)
```

event *Identify events*

Description

Finds the first and last location of an event

Usage

```
event(x, clip = "none", na.rm = FALSE, nas = FALSE, duration = NULL)
```

Arguments

x	A logical statement in which the output is TRUE or FALSE
clip	Defaults to "none", If set to "buffer" a false if placed on each end of the string,if set to TRUE the data is clipped to the first and last time when the event did not occur
na.rm	TRUE/FALSE should NAs be handled
nas	if na.rm is set to true, what should NAs be replaced with TRUE/FALSE
duration	Numeric number selecting the minimum duration of an event

Value

Returns a matix of two columns

start	the location in the original vector at which the event starts
end	the location in the original vector at which the event stops

Author(s)

Connor F. White

See Also

eventInterp

Examples

```
x<-c(.5,.5,.75,.25,.2,.25,.6,.9,.1,.7)
x.e<-event(x<.3)

#Excluding the .1 where there is only 1 value below .3
x.e<-event(x<.3,duration=1)


#If we were interested in values greater than .3 we would need
# to alter the data as the the first and last values are true.

#Starts searching for events from the first time a FALSE occurs
x.e<-event(x>.3,clip=TRUE)

#Starts searching for events from the very beginning
x.e<-event(x>.3,clip="buffer")
```

`eventFunc` *executes function over events*

Description

Executes a funtion over every event and returns the output for each event

Usage

```
eventMean(dat, events, fun=function(x){mean(x,na.rm=TRUE)})
```

Arguments

`dat`	the vector that you wish to take values from
`event`	an output from the event function
`fun`	a function to execute for each event

Value

returns the output of the suppied funtion for each event

Author(s)

Connor F. White

See Also

events

Examples

```
data<-rnorm(100)
on<-rbinom(prob=.3,size=1,n=100)

instances<-event(on==0,clip="buffer")

eventFunc(dat=data,events=instances,fun=function(x){mean(x,na.rm=TRUE)})

eventFunc(dat=data,events=instances,fun=function(x){length(x)})
```

`eventFunc` *executes function over events*

Description

Executes a funtion over every event and returns the output for each event

Usage

```
eventMean(dat, events, fun=function(x){mean(x,na.rm=TRUE)})
```

Arguments

`dat`	the vector that you wish to take values from
`event`	an output from the event function
`fun`	a function to execute for each event

Value

returns the output of the suppied funtion for each event

Author(s)

Connor F. White

See Also

events

Examples

```
data<-rnorm(100)
on<-rbinom(prob=.3,size=1,n=100)

instances<-event(on==0,clip="buffer")

eventFunc(dat=data,events=instances,fun=function(x){mean(x,na.rm=TRUE)})

eventFunc(dat=data,events=instances,fun=function(x){length(x)})
```

eventInterp *Interpret during events*

Description

Identify events such as strings of NA and then linearly interpolating across them

Usage

```
eventInterp(dat, event, clip = "none", na.rm = FALSE, nas = FALSE, duration = NULL)
```

Arguments

`dat`	data set that the interpolated dataset will go into
`event`	A logical statement in which the output is TRUE or FALSE
`clip`	Defaults to "none", If set to "buffer" a false if placed on each end of the string,if set to TRUE the data is clipped to the first and last time when the event did not occur
`na.rm`	TRUE/FALSE should NAs be handled
`nas`	if na.rm is set to true, what should NAs be replaced with TRUE/FALSE
`duration`	Numeric number selecting the minimum duration of an event

Details

See description of event

Value

Returns a vector the same length as the original dataset, yet values are linearly interpolated where the event is true

Author(s)

Connor F. White

See Also

event, eventMean

Examples

```
x<-c(.5,.6,.75,NA,NA,NA,.6,.9,.8,.7)

eventInterp(dat=x,event=is.na(x))
```

`focalfast` *Calculate focal Analysis*

Description

Execute a funciton over a focal window

Usage

```
focalfast(r, window.size, fun = function(x) {
    mean(x, na.rm = TRUE)
})
```

Arguments

`r`	a raster
`window.size`	the number of cells that you would like the focal window to be, must be odd
`fun`	function to calculate in the focal analysis

Value

Returns a raster

Author(s)

Connor F. White and Meghan Blumstein

Examples

```
r<-matrix(rnorm(10000),nrow=100,ncol=100)
r<-raster(r)

map<-focalfast(r=r,windowsize=4,fun=mean)

image(map)
```

`lat2cart` *generate cartesian coordinates*

Description

convert a series of lat and longs to a cartesian coordinate frame centered on an origin

Usage

```
lat2cart(lat, long, latOrg, longOrg)
```

Arguments

`lat`	Latitude(s) to be converted to meters
`long`	Longitude(s) to be converted to meters
`latOrg`	Latitude of origin
`longOrg`	Longitude of origin

Details

returns the x and y location of the set of points in refernce to the origin. Distances are calculated assuming the earth is a sphere.

Value

the output is in x and y coordinates from the origin. If a vector or lats and longs are supplied than a list of x and y coordinates are returned

Author(s)

Connor F. White

See Also

distGeo

Examples

```
#converting one location
lats<-runif(1,-90,90)
longs<-runif(1,-180,180)
latitude_origin<-0
longitude_origin<-0
lat2cart(lats,longs,latitude_origin,longitude_origin)

#Converting many locaitons at once
lats<-runif(10,-90,90)
longs<-runif(10,-180,180)
latitude_origin<-0
longitude_origin<-0
lat2cart(lats,longs,latitude_origin,longitude_origin)
```

`legend.bub` *Add Legend*

Description

To add a legend to a transbub plot, with the bubbles and the counts

Usage

```
legend.bub(Trans, nbub = 3, xper = c(0.6, 0.95), yper = c(0.5, 0.95), sqrt = FALSE, lims.b = NULL,
```

Arguments

`Trans`	Transistion matrix
`nbub`	number of bubles to inclued
`xper`	a vector of length two representing the percent of the x axis to start and the percent of the plot to end the the plotting
`yper`	a vector of length two representing the percent of the y axis to start and the percent of the plot to end the the plotting
`sqrt`	Whether the count sizes should be square root transformed
`lims.b`	the limits to be used when scaling the bubble sizes
`bubsize`	The maximum and minimum bubble sizes
`textl`	The number of text characters to in the label
`yspacing`	list of the spacing within the legend, should be length nbub + 1
`xspacing`	list of the x spacing of the legend, should be length nbub + 1
`boarder`	the spacing to put around the boarders
`rounddig`	digits to round the number to
`pch`	the character type for the bubles
`pt.bg`	color of the background for the point
`pt.col`	color of the point
`bg`	the background color of the legend

Author(s)

Connor F. White

See Also

transBub

Examples

```
rec<-sample("A","B","C",size=200)
trans<-transMat(rec)

transBub(trans)
legend.bub(trans,nbub=3)
```

`legend.Bubble` *Generating legend*

Description

Adding a legend onto a plot

Usage

```
legend.Bubble(text, pt.cex, textl = NULL, xper = c(0.6, 0.95), yper = c(0.5, 0.95), yspacing = NULL
```

Arguments

`text`	the string for labels
`pt.cex`	the size of the bubbles
`nbub`	number of bubles to inclued
`xper`	a vector of length two representing the percent of the x axis to start and the percent of the plot to end the the plotting
`yper`	a vector of length two representing the percent of the y axis to start and the percent of the plot to end the the plotting
`sqrt`	Whether the count sizes should be square root transformed
`lims.b`	the limits to be used when scaling the bubble sizes
`bubsize`	The maximum and minimum bubble sizes
`textl`	The number of text characters to in the label
`yspacing`	list of the spacing within the legend, should be length nbub + 1
`xspacing`	list of the x spacing of the legend, should be length nbub + 1
`boarder`	the spacing to put around the boarders
`rounddig`	digits to round the number to
`pch`	the character type for the bubles
`pt.bg`	color of the background for the point
`pt.col`	color of the point
`bg`	the background color of the legend

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (text, pt.cex, textl = NULL, xper = c(0.6, 0.95), yper = c(0.5,
    0.95), yspacing = NULL, xspacing = NULL, boarder = 1, pch = 21,
    pt.bg = "blue", pt.col = "black", bg = "blue")
{
    nbub <- length(pt.cex)
    if (is.null(textl)) {
        textl <- max(sapply(text, nchar))
    }
    if (is.null(yspacing)) {
        bs <- pt.cex/2
        yspacing <- c(bs, boarder)
        yspacing[2:length(yspacing)] <- yspacing[2:length(yspacing)] +
            bs
        yspacing[1] <- yspacing[1] + boarder
        yspacing <- cumsum(yspacing)/sum(yspacing)
        yspacing <- yspacing[1:nbub]
    }
    if (is.null(xspacing)) {
        xbub <- c(max(pt.cex))
        bs <- xbub/2
        xspacing <- c(xbub, textl, boarder)
        xspacing[2:length(xspacing)] <- xspacing[2:length(xspacing)] +
            bs
        xspacing[1] <- xspacing[1] + boarder
        xspacing <- cumsum(xspacing)/sum(xspacing)
        xspacing <- xspacing[1:2]
    }
    coords <- par("usr")
    xdif <- (coords[2] - coords[1])
    ydif <- (coords[4] - coords[3])
    xper <- xper * xdif + coords[1]
    yper <- yper * ydif + coords[3]
    ycoord <- yper[1] + (yper[2] - yper[1]) * yspacing
    xcoord <- xper[1] + (xper[2] - xper[1]) * xspacing
    rect(xper[1], yper[1], xper[2], yper[2], col = bg)
    points(x = rep(xcoord[1], nbub), y = ycoord, cex = pt.cex,
        pch = pch, bg = pt.bg, col = pt.col)
    text(x = rep(xcoord[2], nbub), y = ycoord, labels = text,
        pos = 1, offset = -0.5)
}
```

`legend.line` *Add line width legend*

Description

a wrapper for the legend command to place the line sizes onto a transBub plot

Usage

```
legend.line(Trans, x = NULL, y = NULL, nline = 4, lims.l = NULL, linesize = c(1, 5), sqrt = FALSE, r
```

Arguments

`Trans`	Transistion matrix full of count data
`x`	x coordinate of the plot
`y`	y coordinate of the plot
`nline`	number of lines to have in plot
`lims.l`	vector of length two representing the min and max counts
`linesize`	vector of length two representing min and max line sizes
`sqrt`	should the count data be square root transformed
`rounddig`	number of digits to round numbers to
`bg`	color of background

Author(s)

Connor F. White

See Also

legend.bub, transbub

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (Trans, x = NULL, y = NULL, nline = 4, lims.l = NULL,
    linesize = c(1, 5), sqrt = FALSE, rounddig = 2, bg = "White")
{
    if (is.null(x) | is.null(y)) {
        coords <- par("usr")
        x <- coords[1]
        y <- coords[4]
    }
    counts <- totcount(Trans)
    bubCounts <- counts$TransCount
    if (is.null(lims.l)) {
        llab <- seq(min(bubCounts[bubCounts > 0]), max(bubCounts[bubCounts >
            0]), length.out = nline)
    }
    else {
        llab <- seq(lims.l[1], lims.l[2], length.out = nline)
    }
    lsize <- scal(llab, scal = linesize, lims = lims.l, sqrt = sqrt)
    legend(x = x, y = y, legend = round(llab, digits = rounddig),
        lty = c(1, 1, 1, 1), lwd = lsize, bg = bg)
  }
```

`locMap` *generation a location dataframme*

Description

based on a series of states, and locations, it will generate a summarized matrix listing the lcoation of each state. Designed to generate the location of all the stations from a VEMCO detection history

Usage

```
locMap(state_rec, Lats, Longs)
```

Arguments

`state_rec`	vector of the states
`Lats`	vector of y locations
`Longs`	vector of x locations

Value

returns a matrix with three columns. Column one is station name, column two is the y coordinate, and column three is the x coordinate

Author(s)

Connor F. White

See Also

transBub

Examples

```
rec<-sample("A","B","C",size=200)
lats<-rec
long<-rec

lats[lats=="A"]<- 33
lats[lats=="B"]<- 34
lats[lats=="C"]<- 31

long[long=="A"]<- -118
long[long=="B"]<- 120
long[long=="C"]<- -117

locs<-locMap(state_rec,lats,long)
```

`locsloc` *list locations*

Description

create a list of all possible combination of rows

Usage

```
locsloc(locs)
```

Arguments

`locs`	list of locations

Details

internal function used for transition matrix things

Value

martix with two columns

Author(s)

Connor F. White

Examples

```
mat<-matrix(1:10,nrow=10)
locsloc(mat)
```

`logGen` *Generate a state summarization*

Description

turn a time series of state data into a time standardized summary log file. Creates a matrix where each row is a time step and each column represents the number of a state occured within that time period

Usage

```
logGen(state_rec, times, Time_Step, start, end, states = NULL)
```

Arguments

`state_rec`	vector of states
`times`	POSIX vector representing the time that each state was recorded
`Time_Step`	the number of seconds that the data will be binned over
`start`	POSIX item refering to when the log file will start
`end`	POSIX item refering to when the log file will end
`states`	a vector listing all the states to be included in the log file

Value

A matrix with each column representing a single state. The number of columns will default to to the number of unique states, unless the states argument is supplied. Each row represents a time step, with the number of rows being dependent on the start and end argument. Each cell represents the number of times that that state occured between t and t+1.

Originally designed to determine the number of detections on acoustic receivers over time.

Author(s)

Connor F. White

Examples

```
Station<-sample("A","B","C",size=200)
start<-as.POSIXct("2016-04-26")
end<-start<-as.POSIXct("2016-04-28")
time<-seq(start,end,length.out=50)
time<-rep(time,4)
time<-time+rnorm(200,mean=3600)
```

```
#Generate a log file every hour
logGen(state_rec=station,times=time,Time_Step=3600,start=start,end=end)

#generate a log file ever 5 mins
logGen(state_rec=station,times=time,Time_Step=300,start=start,end=end)

#supply a list of states to possibly standardize, even if the state never offccured in this example

logGen(state_rec=station,times=time,Time_Step=3600,start=start,end=end,states=c("A","B","C","D"))
```

`plot.tran` *Plot Transitions*

Description

Plot transistions between all points, which there is an lwd

Usage

```
plot.tran(locs, lwd = 1, col = "black", bend = 0.05, head = 0.1)
```

Arguments

`locs`	A three column matrix or dataframe. column one is the station names in the same order as rownames in Trans. The second column is the y coordinates, and column three is the x coordinates.
`lwd`	the line width of each transistion. Should be length nrow(locs)^2
`col`	color of the transitions
`bend`	the amount to bend the transition lines. The porportion of distance between each node that the line will be maximumly away from the straight path. If 0 then lines are not bent
`head`	the length in inches of the head of the arrows, if 0 then lines are drawn with no arrow heads

Details

This is used in transBub, however, if you only want to view transistions or edges that this function can be used. However, you must supply the list of linewidths yourself. To determine the order examine locsloc or can calculate the number of counts using totcount command

Author(s)

Connor F. White

See Also

transBub, transmat, totcount, locsloc

Examples

```
locs<-circLocs(5)
dat<-sample(c("A","B","C","D"),100,replace=TRUE)
x<-transMat(dat, States=c("A","B","C","D"))
counts<-totcount(x)
lwds<-counts$Transcount
lwds[lwds>0]<-scale(lwds[lwds>0],scal=c(1,5))

plot(locs[,2]~locs[,3],type="n")
plot.tran(locs,lwds=lwds)
```

`scal` *Scale a vector*

Description

This is designed to rescale a vector, between two numbers. often designed for plotting purposes to scale valeus for point sizes

Usage

```
scal(x, scal = c(0, 1), lims = NULL, sqrt = F, na.rm = F, limit = TRUE)
```

Arguments

`x`	numbers to be rescaled
`scal`	vector of two listing the new minimum and maximum values
`lims`	values that should represent the minimum and maximum scaled value
`sqrt`	should the data be square root transformed
`na.rm`	how to handle NAs
`limit`	TRUE/FALSE if lims is supplied, should values smaller than the minimum lims or greater than the maximum lims be limited to the minimum and maximum scal

Value

a vector of rescaled values

Author(s)

Connor F. White

Examples

```
values<-rnorm(100)

#rescale the values between 0 and 1
scal(values,scal=c(0,1))

#in order to minimize influence of outliers
scal(values,scal=c(0,1), lims=c(-1,1))

#allow values outside of -1 and 1 to get bigger and smaller
scal(values,scal=c(0,1), lims=c(-1,1),limit=FALSE)
```

`sunCalc` *Sunrise and sunset time*

Description

given the latitude and longitude of any day calculate the time of sunrise and sunset. Aquired from..... the internet

Usage

```
sunCalc(d, lat, long)
```

Arguments

`d`	POSIX item representing the day of interest
`lat`	Latitude of location in decimal degrees
`long`	Longitude of location in decimal degrees

Value

returns a list with two levels

`sunrise`	a vector in the list that is composed of the hour of the day in decimal hours that the sun will rise
`sunset`	a vector in the list that is composed of the hour of the day in decimal hours that the sun will set

Author(s)

Connor F. White

Examples

```
day<-as.POSIXct("2016-01-01")
suncalc(day,lat=33.5,long=-118.5)

#if we move the location
suncalc(day,lat=40.5,long=-115)


#Or the values for the entire year
end<-as.POSIXct("2016-12-31")
days<-seq(day,end,by="day")
suncalc(d=days,lat=33.5,long=-118.5)
```

`totcount` *Summarize Transistion matrix*

Description

Summarize a transistion matrix to determine the number of times each state occured and the edges, which is the number of times the state changed

Usage

```
totcount(Trans)
```

Arguments

`Trans`	n x n Transistion matrix

Value

returns a list with two levels

`StateCount`	the number of times each state occurs
`TransCount`	the number of times each transition occurs, the diagonal, which represents a transistion onto itself is 0

Author(s)

Connor F. White

See Also

transMat

Examples

```
dat<-sample(c("A","B","C","D"),100,replace=TRUE)
x<-transMat(dat, States=c("A","B","C","D"))
counts<-totcount(x)

counts$StateCount
counts$TransCount
```

---

transBub *Plot Transistion Matrix*

Description

plot a transition matrix, from transmat. Locations can be supplied for the nodes, however, if not then the nodes are aligned in a circle.

Usage

```
transBub(Trans, locs = NULL, bubsize = c(1, 5), lims.b = NULL, bub.pch = 21, bub.col = "black", bub
```

Arguments

`Trans`	a n x n matrix with count data representing the number of transistions from each state to each other states
`locs`	A three column matrix or dataframe. column one is the station names in the same order as rownames in Trans. The second column is the y coordinates, and column three is the x coordinates. if not provided data locations will be provided in a sphere
`bubsize`	vector of length two representing the cex parameter of the minimum and maximum node locations
`lims.b`	if supplied a vector of length two representing the count data representing the minimum and maximum bub sizes, else defaults to minimum and maximum observed in the data
`bub.pch`	Character type of the nodes
`bub.col`	color of the nodes
`bub.bg`	color of the nodes background
`bubsqrt`	should the count data for the nodes be square root transformed
`linesize`	minimum and maximum line widths
`lims.l`	if supplied a vector of length two representing the count data representing the minimum and maximum line widths, else defaults to minimum and maximum observed in the data
`line.col`	color of the lines
`bend`	the amount to bend the transition lines. The porportion of distance between each node that the line will be maximumly away from the straight path. If 0 then lines are not bent
`head`	the length in inches of the head of the arrows, if 0 then lines are drawn with no arrow heads

`linesqrt`	should the transistion counts be square root transformed
`add`	should theis plot be added onto the current plotting surface. If set to true, recommended that you have supplied locs
`xlim`	vector of length two representing the minimum and maximum x coordinates of the plot
`ylim`	vector of length two representing the minimum and maximum y coordinates of the plot

Details

Can be plotted independantly. However, when locs are supplied can be added onto a map with add = true. Designed for representing these fish movements from passive acoustic telemetry

Author(s)

Connor F. White

See Also

transMat, legend.bub, legend.line

Examples

```
dat<-sample(c("A","B","C","D"),100,replace=TRUE)
x<-transMat(dat, States=c("A","B","C","D"))

transBub(x)

#If we want to supply locations for each node
locs<-data.frame(State=c("A","B","C","D"),y=rep(0,4),x=c(1:4))
transBub(x,locs=locs)

#If we want to supply the minimum and maximum values of the nodes
transBub(x,lims.b=c(0,40),lims.l=c(0,10))
```

`transMat` *Transition counts*

Description

Create counts in the form of a matrix from one state to the next state.

Usage

```
transMat(State_rec, prob = FALSE, States = NULL)
```

Arguments

`State_rec`	This is a list of the currents states can be in numeric, character or factor form
`prob`	This is a item that when set to true return ths probabilities instead of the counts in each cell
`States`	This is a item that when given is the list of states that you would like to use, Otherwise defaults to unique(State_rec)

Details

for plotting see transBub

Value

Returns an n by n matrix, with n being the number of states. If states is not supplied, defaults to the number of unique values in State_rec. If prob is true, then these counts are transformed into probabilities, with the sum of each column equalling 1. The counts in each cell are based on the number of times a transistion from the state represented by the column to the state represented by the row occured. See transBub for plotting

Author(s)

Connor F. White

See Also

transBub

Examples

```
#make 100 values of blue green and yellow

dat<-sample(c("A","B","C"),100,replace=TRUE)
transMat(dat)
transMat(dat,prob=TRUE)
```

Index